福井大学
工学部　研究報告
第40巻　第1号
1992年3月

分散型関係ＤＢＭＳの設計と構築

都司達夫[1]　丸山正理[2]　大木下俊也[3]　冨士竹仁[4]　田中仁士[4]
上坂利文[5]　加藤昌央[6]　木本　茂[7]　林　利治[8]　渡辺勝正[1]

Design and Implementation of Distributed Relational DBMS

Tatsuo TSUJI　Masari MARUYAMA　Toshiya OGINOSHITA
Takehito FUJI　Hitoshi TANAKA　Toshifumi UESAKA　Masao KATO
Shigeru KIMOTO　Toshiharu HAYASHI　Katsumasa WATANABE
(Received Feb. 8, 1992)

This paper describes a distributed relational database management system with emphasis on design and implementation. We have designed and constructed the system on the UNIX local area network in our university. It is based upon the SQL standard, and distributed processings are implemented using RPC (Remote Procedure Calls). The main features of the system include:

(1) Multi-client and multi-server system,
(2) Client-based distribution management,
(3) Deadlock free concurrency control scheme,
(4) Client-controlled load distribution scheme.

It should be noted that the current version of the system does not support the transaction processing and access security control.

1．緒　　言

データベース管理システム(DBMS)は技術集約型・知識集約型の大規模システムである。オペレーティングシステムに深く関わったレベルから，データ構造・言語処理，ユーザインターフェースに至るまで，ありとあらゆる技術要素がそこには含まれている。優れたDBMSを構築するためには，計算機科学の諸分野における最先端の技術や知識を総動員する必要があるといえる。

本稿は，UNIXネットワーク上における分散型関係DBMSの設計と構築に関する研究内容をまとめたものである。なお，本分散DBMS(以下，簡単のためにD-SQLと呼ぶ)は関係データベース[2][6]の世界的な標準となっているSQL (Structured Query Language)[1]に準拠しており，NFS (Network File System) において実装されているRPC (Remote Procedure Call - 遠隔手続き呼び出し)[4][5][9]を用いて分散化を実現している。

---

1 情報工学科　2 丸山メガネ㈱　3 三谷商事㈱　4 日立製作所㈱
5 日本電気㈱　6 島津製作所㈱　7 北陸日本電気ソフトウエア㈱　8 松下寿電子工業㈱

D-SQL の特徴は以下の4点にまとめることができる。

(1) マルチクライアント・マルチサーバシステムである。

ネットワーク上の動作形態としてはD-SQLはクライアント・サーバ方式により動作する。分割して存在するデータが各々の計算機上に唯一つ存在するサーバによって管理されるため本システムはマルチサーバで動作する。また1人のユーザに対して1つのクライアントがその要求を受け付ける。従ってマルチユーザを支援するためには，マルチクライアントで動作する必要がある。すなわち本システムはマルチクライアント・マルチサーバ方式のシステムである(図1)。なお，データベースの分散の粒度はテーブル単位の分散である。

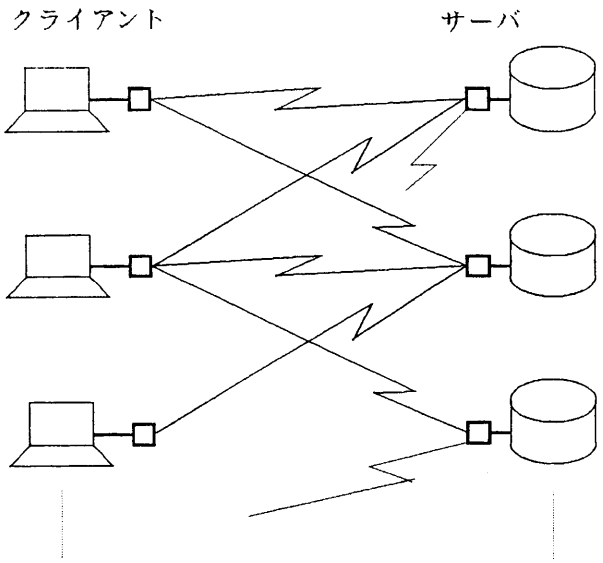

図1　マルチクライアント・マルチサーバシステム

(2) クライアント主導の分散管理方式である。

D-SQL の分散管理はクライアント側のカーネルによって行なわれる。クライアントが分散データベースにコネクトした時に主データベースのシステムカタログからデータベースの分散情報を収集しクライアント側の内部データ構造に変換する。この内部データ構造により，テーブルの論理名に対して，その物理名とネットワークにおける位置が対応付けられ分散透過性が実現される。サーバ側の分散情報の原本と，このクライアント側のレプリカの一貫性を保証する方策が必要であるが，これについては本論で述べる。

(3) デッドロックフリーな同時実行制御

D-SQL の排他制御はテーブル単位で行なわれるが，これはテーブルにロックをかけることによって行なわれる。ロックの種類はテーブル読み出し時にかける共有ロックと更新時にかける排他ロックがあるが，ロッキングによりデッドロックが起こり得る。一般に分散環境下のデータベースのデッドロックの検出と回復の問題は非常に困難な要素を含むが，D-SQL においてはデッドロック発生の可能性のない単純な方策を採用した。

(4) クライアント主導の負荷分散方式である。

ネットワーク上の分散データベース管理システムを設計する上での主な要点は以下の2点である。

(i) サーバノード(群)への負荷集中の軽減

(ii) ネットワーク通信量の軽減

以上の2点を共に満足する解決法を見つけだすことは非常に困難であるし，データベースの運用形態も多様であることを考えると唯一絶対的な方策というようなものはあり得ない。D-SQLでは(i)の点を重視した上で，可能な限りネットワーク通信量を軽減するような方策とする。そのために，データベースの中心的な処理である条件探索操作について，負荷の比較的軽いものだけをサーバで行い，テーブルの結合操作(join)などを含む負荷の重い条件探索操作はクライアント側で行うこととする。これによって，サーバノードに対する負荷は，かなりクライアント側に移動することとなり，システム全体としてのスループットを向上させることができる。質問処理の最適化の手順も含めて，データベース処理をサーバ側で行なうかクライアント側で行なうかの負荷分散のプランはクライアント側で立てることにする。

本稿では，D-SQLの全体の設計方針や，サーバ側およびクライアント側のカーネル部分を中心に述べる。SQLコマンドインタープリタや全体にわたる詳しい説明は文献[10]を参照されたい。

2．システムテーブルによるデータベース管理

本節では，D-SQL全体のモジュール構成を説明した後，システムテーブルによるサーバ側のデータベース管理について述べる。

2.1 D-SQLのモジュール構成

D-SQLは，大きく分けて論理的に2つの部分からなる。データベースにアクセスして所定の処理を行うデータベース操作ライブラリ(サーバ側)と，コマンドを解析した後，必要な操作ライブラリルーチンを呼び出して実行するコマンドインタープリタ部である(クライアント側)。図2にD-SQL全体のモジュール構成を示す。

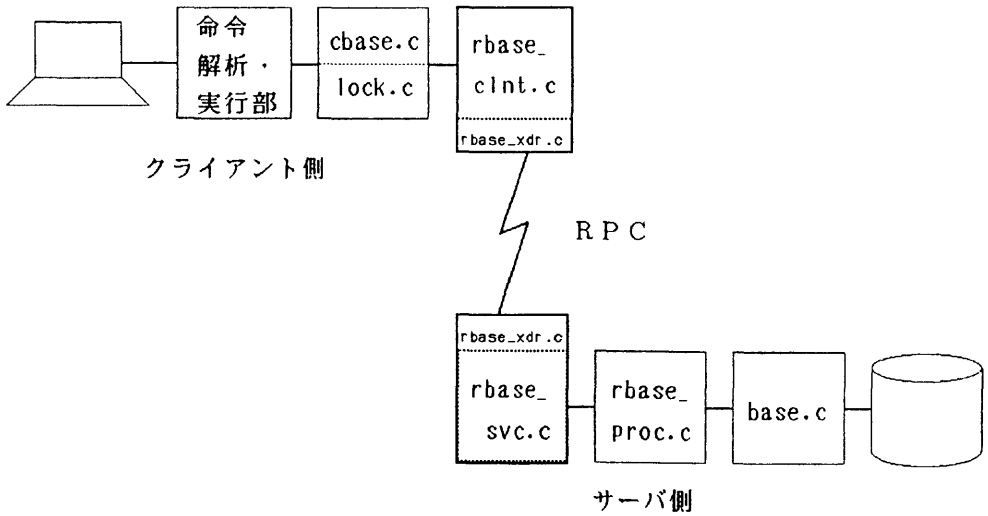

図2 D-SQLのモジュール構成

(1) サーバ側モジュール構成

base.c

B⁺木操作ライブラリを用いて最も低レベルなデータベース操作を行う。主な操作は,

- レコードの検索, 更新, 削除, 挿入
- データベースの作成
- テーブルの作成, 削除

rbase_proc.c

クライアントから呼び出された関数の引数の変換, および排他制御を行う。クライアントからサーバへの要求は, RPC の関数呼び出しの形で行われるが, この関数に対する引数の指定の仕方は RPC の仕様に従っている。クライアントから 関数呼び出しがあると, 対応する rbase_proc.c の関数が実行される。rbase_proc.c の関数に渡された引数を base.c の関数の引数に変換した後, それを呼び出す。データベース操作を行う関数を呼び出す他, ネットワークの設定や制御に関する関数もこのモジュールに含まれる。

rbase_xdr.c

ネットワーク上のデータフォーマットと計算機内部でのデータ表現の変換を行うルーチン群(サーバ, クライアント共通)。

rbase_svc.c

クライアントからの呼び出しを直接受け付けるルーチン群。サーバ側で, 最もネットワーク側にあるモジュールである。クライアントの呼び出しを受けると, rbase_xdr.c 中のルーチンを呼び出し引数を変換する。また, クライアントによって指定された関数を実行して結果をネットワークを通じてクライアントに返す。

(2) クライアント側モジュール構成

rbase_clnt.c

サーバの関数呼び出しを行うルーチン群。クライアント側で, 最もネットワーク側に近いモジュールであり, rbase_xdr.c 中のルーチンを呼び出し引数の変換を行う。

cbase.c

ネットワーク透過性や分散透過性を保証するためのモジュール。

主な役割は,

(a) サーバとの間の通信路の設定

(b) サーバの関数呼び出しの引数, 戻り値の変換

(c) 分散管理

である。(b)はサーバ側における rbase_proc.c の対応機能に相当し, cbase.c より上位のモジュールから受け取った引数を, rbase_clnt.c での引数に変換する。

lock.c

サーバに対して, 排他制御の要求を出す。

SQL 命令解析・実行部

SQL のコマンドインタープリタ。

2.2 データベース管理の方針

データベースシステムの管理の基本的な対象はテーブルのカラムである。従って，場合によっては数千にも及ぶようなカラムをいかにして，分かりやすく効率的に管理するかということが設計上のもっとも大きな問題である。

我々は，このカラムの管理のための構造について2，3のプロトタイプを作った後システムテーブルと呼んでいるデータ構造を採用した。ユーザの要求に基づいて，新たなカラムの登録や削除は頻繁に起こる。すなわち，カラムの管理構造は基本的に動的なものである必要がある。このような要求を満たすために低レベルデータ構造としてB⁺木[3]を使用するのが適切であり，自然で効率よい管理の方法であるといえる。

この考えを一歩進めて，ユーザがアプリケーションで定義するSQLのテーブル(以後，ユーザテーブルと呼ぶことにする)とまったく同じ構造を持つSQLテーブルでカラムの属性や関連情報を管理しようというのがシステムテーブルの発想である。カラム情報をSQLテーブルそのもので管理することから得られる利点は絶大であり，上記の要求をシステムの核の部分で満足するにはうってつけの方法であるといえる。利点を挙げると，

(1) ユーザはシステムテーブルに対して，SQLのselectコマンドを使用して，カラム定義情報やテーブル定義情報を知ることができる。従って，selectコマンドの多彩な検索機能を利用して，種々の角度からデータベース情報を知ることができる。

(2) データベース管理のプログラムそのものが分かり易くなる。システムテーブルを操作するのにselect，insert，deleteなどのユーザテーブルを操作するためのSQLコマンドを支援するルーチンの大部分が共用できるからである。

(3) カラムやテーブルについて，その定義情報の他に新たな管理情報を付加することが容易である。

2.3 システムテーブルによるカラムの管理

システムテーブルの構成を図3に示す。なおシステムテーブルの各カラムはすべてキーカラムになっており，B⁺木による索引が与えられている。システムテーブルをSQL構文で定義すると次のようになる。実際はシステムテーブルのカラム数は現在9つである。他の3つは分散管理情報として必要になるものであるがこれについては3節で説明する。

```
create table sys(
        uname   char(20)   not null unique,    -- ユーザ名
        tname   char(20)   not null,           -- テーブル名
        cname   char(20)   not null,           -- カラム名
        type    char(10)   not null,           -- カラムのデータ型
        idx     char(10)   not null,           -- インデックスの種類
        offset  long       not null unique);   -- 構造体COLへのポインタ
```

- 'uname'：テーブルを作成したユーザの名前，すなわちテーブルの所有者名を示す。このカラムは重複を許さないユニークキーである。
- 'tname'：ユーザテーブルの名前を示す。
- 'cname'：ユーザテーブルのカラム名を示す。

・'type'：カラムのデータ型を示す。文字列型の可変長カラムもサポートしている。ただし，可変長カラムをキーにすることはできない。
・'idx'：カラムがキーであるかどうかをまたキーであるならば重複を許すのかどうかを示す。
・'offset'：カラム情報を格納する構造体(COL)へのポインタである。キーカラムであれば，COLのメンバhdはB⁺木(索引)の情報を格納する構造体(TREE)へのポインタを保持する。

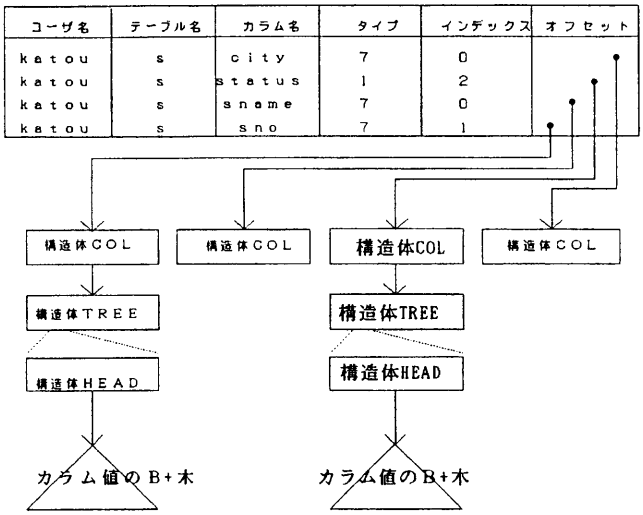

図3　システムテーブルの構成

3．分散管理

通常，分散DBMSとは位置的に分散した複数のデータベースを通信ネットワークを介して，ユーザにはあたかも単一のデータベースとしての見方と取扱を可能にするDBMSであると定義される[8]。分散DBMSを設計するときの重要でかつ興味あるポイントは，すっきりした統一的な管理の下に，いかにしてシステム全体の効率を落とさずにこのことを可能にするかということであるといえる。本節ではD-SQLにおける分散管理の方法について述べる。分散データベースは1つの主データベースと0個以上の副データベースからなっており，主データベースのシステムテーブルは分散データベース全体の分散情報を管理している。また，副データベースには，当該分散データベースに属しているものと，他の分散データベースに属しているものとの2種類ある。分散化のためのプログラミングには，RPCを用いている。

3.1　分散データベース

理解の混乱を防ぐために，ここでは分散データベースや分散DBMSの構成要素を指示する用語を定義する。

データベース

テーブルの集合。一つの.idxファイルと一つの.datファイルに格納される。

システムテーブル

データベースに含まれるテーブルの一つであり，ユーザテーブルの情報やそれらのカラムの情報を管理する。データベース中に必ず一つだけ存在する。

ノ ー ド

ネットワーク上の一つの計算機。複数のデータベースを格納することができる。

サ ー バ

一つのノード内のすべてのデータベースを管理するソフトウエア。

分散データベース

一つの主データベースと0個以上の副データベースから構成される。

主データベース

分散データベース中の全てのテーブルとカラムの情報を格納するシステムテーブルを含むデータベース。他の分散データベースのクライアントはその中にテーブルを定義し作成することはできない。しかし，その中に定義されているテーブルに対して，link コマンドにより，その使用を宣言した後，使用することが出来る。

副データベース

分割副データベースまたは結合副データベース。一般には他のノードに存在し，従ってそのノードのサーバの管理下にある。主データベースと同じノード上にあっても構わない。

分割副データベース

当該分散データベースに所属するデータベース。他の分散データベースのクライアントはその中にテーブルを定義し作成することはできない。しかし，その中に定義されているテーブルに対して，link コマンドにより，その使用を宣言した後，使用することが出来る。

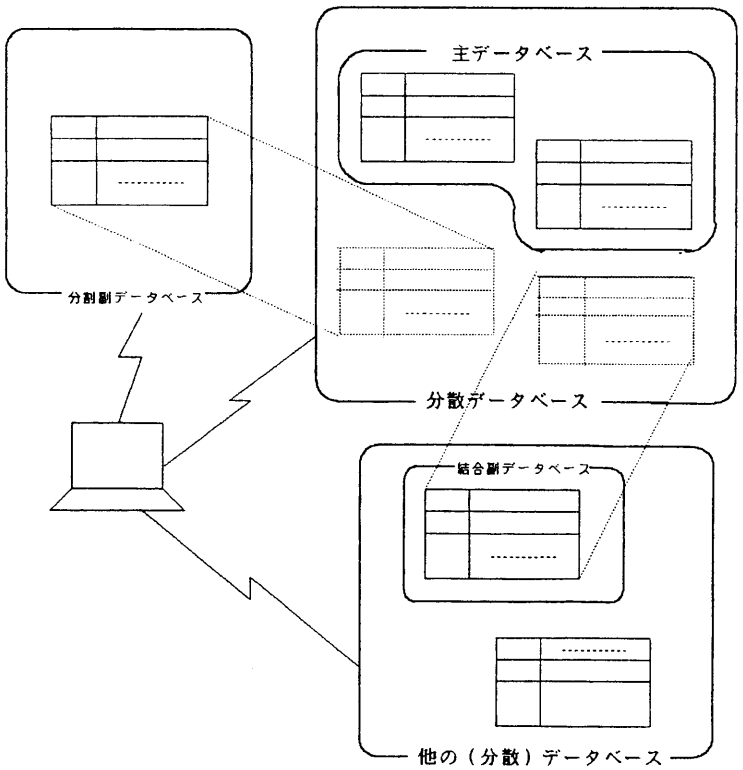

図4　分散データベース

結合副データベース

他の分散データベースに所属するデータベース中のテーブルの集合。link コマンドでそのテーブルの使用を宣言した後，使用できる。

図4を参照されたい。

以上の定義より，一つの分散データベースは一般に複数個のノード上に存在し，それぞれのノードにおけるサーバにより分担して管理される。

クライアント

分散データベースを管理する一般には複数のサーバと交信して，所定のデータベース処理を依頼するソフトウエア。一つの分散データベースに対して，複数個のクライアントが同時に交信できる。

従って，分散データベース管理システムはマルチクライアント・マルチサーバシステムである。また分散データベースの特定の形態として，

集中データベース

主データベースのみからなる分散データベース。

3.2 分散データベースの透過性

データベースの透過性とはユーザにとって，またはDBMSのより上位レベルのモジュールにおいて，データベースが論理的に単一のものとみなすことができ，取り扱うことができるという性質である。格納媒体，通信媒体，所在などの物理的要素をできるだけ意識させずに操作できることが必要である。分散データベースの場合次の2つの透過性が重要である。

(1) ネットワーク透過性

分散データベース中の各データベースが実際にはネットワークを介して結合されていても，ネットワークの介在を意識させないで操作できる性質である。

RPCではサーバ側の登録ルーチンの引数，及び結果は構造体ではなくそれを示すポインタが返され，さらに登録ルーチン名に'_'とバージョン番号が付加されるので必ずしも使いやすいとはいえない。そこで，ここでは使いやすく，拡張性のあるDBMSを構築するために，コマンドインタプリタ(isql)とクライアントスタブとの間にRPCとのインタフェースを行う部分を2.1節で述べたモジュールcbase.cの中に設けている。このインタフェース部分が間に介入することによって，サーバ側のモジュールbase.c中のデータベース操作ルーチンをRPCやネットワークを意識しないで，あたかも直接呼び出しているように，引数や結果も何の変更もなく受け渡しが可能となる。

(2) 分散透過性

分散データベース中の各データベースが実際にはどの計算機ノードにあるのか，あるいはテーブルがどのノードのどのデータベース中にあるのかといった物理的な所在を指定しなくてもよいようにすること。このために種々の物理的な指定と論理的な指定を相互変換する必要があるがこれについては次節以降で述べる。

3.3 分散管理の方法

2.3節で述べたシステムテーブルは基本的に"集中データベース"のためのものであり，ユーザテーブルの管理情報として，テーブル作成者とテーブル名が登録されていた。分散データベース中のテーブルについてはこれだけの情報ではテーブルの所在を同定できない。他にテーブルが存在している

ノード名とデータベース名(主データベース名または副データベース名)の情報が必要になる。すなわち，テーブルの所在を指定するために，

（ノード名，データベース名，テーブル名） ①

の３つの情報が必要である。従って，システムテーブルに“ノード名”と“データベース名”を格納するための新たなカラムを増設しなければならない。

もちろん，ユーザにこのような物理的な所在を意識させてテーブルを操作させるわけではなく，これらの所在情報に論理的な名前を付与して論理名でテーブルを操作できるようにしなければならない。これをテーブルの論理名といい，この論理名で示されるテーブルを“論理テーブル”と言う。これに対して，①におけるテーブル名を物理名と言い，物理名で示されるテーブルを“物理テーブル”という。テーブルが主データベース内または分割副データベース内にあるときにはテーブルの論理名と物理名は同一であり，また，結合副データベース内にあるときには，論理名と物理名は異なっていてもよいし，同一であってもよい。結合副データベース内にあるとき，物理名は結合副データベース内のシステムテーブル中の論理テーブル名と同一である。

なお，テーブル作成者の情報(uname)は所在情報としては使われることはない。この情報は，テーブル操作の権限チェックのために使われる。

結局，分散データベース中のテーブルカラムを同定するために，３つのカラムが増設されることになり，システムテーブルのカラムは以下の９つに拡張される。DUP はキーの値が重複してもよいことを示す。

カラム名	データ型	インデックスの型	
uname	CHAR	DUP	
tname	CHAR	DUP	論理テーブル名
cname	CHAR	DUP	
idx	CHAR	DUP	
type	CHAR	DUP	
offset	LONG	DUP	
hname	CHAR	DUP	ノード名
dbname	CHAR	DUP	データベース名
ptname	CHAR	DUP	物理テーブル名

3.4 クライアント側における集中分散制御

システムテーブルに新たに付け加えたカラムデータは，ユーザが指定した論理テーブル名を，物理テーブルの所在情報(ノード名，データベース名，物理テーブル名)に変換するために使用される。この所在情報への変換は，テーブルへのアクセスがある度に，頻繁に行われるが，システムテーブルの検索はディスクアクセスを伴うため，速度が遅い。そこで，変換のために必要な情報を，クライアントのメモリ上にコピーし，クライアント側で変換を行うことにする。

すなわち，クライアントはサーバからシステムテーブル中の所在情報に関係する部分のみをコピーし，この情報にしたがって，分散管理のためのデータ構造を作成する。分散制御は，これらのデータ構造を使用して，クライアント側において集中的に行われる。所在情報のコピーとデータ構造の作成は関連するすべての副データベースについて RPC コール connect_dbase() の中で行われる。

このことで，変換のたびに当該サーバを呼び出し，システムテーブルを検索するオーバヘッドを大幅に減少させることができる。ただし，サーバ側の実際のシステムテーブルとの一貫性が保たれなければならず，そのためのオーバヘッドを伴う。しかし，このオーバヘッドはディスク中のサーバ側のシステムテーブルをその都度，検索するオーバヘッドに比べれば問題にはならない。

3.5 分散管理情報の一貫性

分散管理をクライアント側で集中して行うために，各々のデータベースのシステムテーブル中のテーブルの所在情報をクライアント側にコピーすることを前節で述べた。この一貫性は，サーバ側のシステムテーブルの更新によって失われる。システムテーブル中のユーザテーブル情報の更新は新しくユーザテーブルを生成したり消去したりする cbase.c 中の関数 def_column()，drop_table() および参照副データベース中のテーブルにリンクしたりアンリンクしたりする関数 tbl_link()，tbl_unlink() を呼んだ時に起きる。従って一貫性を保持するために，これらのルーチンが呼ばれシステムテーブルが更新された場合には，これを検出して新しいシステムテーブルをサーバ側から再度コピーして，内部データ構造を再構成する必要がある。

システムテーブルが更新されたことを検出するために，システムテーブルの更新回数を数えるカウンタをサーバ側に設ける。このカウンタは，システムテーブルの更新の度に 1 加算される。クライアントは，システムテーブルのコピーをとる時に，同時にこのカウンタの値を記憶しておく。そして，システムテーブルをアクセスする度にこのカウンタ値を照合し，更新されていないことを確かめる。クライアントにコピーしたカウンタ値と，サーバ側の実際のカウンタ値が等しくなければ，システムテーブルを再度コピーする。

このような検出方法をとった理由はサーバとクライアント間の通信が RPC コールで行われることに起因する。RPC コールは関数呼び出しであるから，“半 2 重通信”と考えることができる。そして，クライアントがサーバを呼び出すのであるから，送信権はクライアントにある。よって，サーバが，システムテーブルが更新されたことを非同期にクライアントに知らせることができない。従って，システムテーブルの更新の確認は，送信権をもつクライアント主導で行わなければならない。

4．同時実行制御

多くのユーザが 1 つのデータベースを同時に使用しようとする場合データの一貫性を損なわないようにするために同時実行制御が必要である。同時実行制御に伴う重要な問題としてデッドロックの問題があるが，これは本システムのような分散システムの場合には難しい問題を含む。

分散制御がクライアント側のモジュール cbase.c において，集中して行われたのに対して，同時実行制御の場合には，クライアントからの排他制御の要求を受け付けたサーバ側のモジュールは排他制御のためのデータ構造を作成し，それにより同時実行制御を行うというような協調動作が行われる。

4.1 同時実行制御の必要性

D-SQL では，サーバ・クライアント間の通信に RPC を用いている。ある RPC ルーチンを実行している間は，他の RPC ルーチンを実行できない。すなわち，一連の RPC ルーチンの実行は常に逐次的であり，従って，RPC ルーチン単位では，RPC のメカニズムにより処理が排他的に行われている。ところが，RPC ルーチン単位の処理の排他性が保証されていてもデータベースの排他

制御は十分ではない。１つのSQLコマンドは一般に複数個のRPCルーチンの呼び出しにより実行されるからである。

(1) 同時実行制御の時間範囲と対象データの範囲

同時実行制御の時間範囲は，ひとつのSQL操作文の実行が行われる間とする。データ操作は，すべてSQL操作文を単位として行われる。そこで，ここでは操作文の処理を開始する直前から，処理の終了までを同時実行制御の時間範囲とする。このことの妥当性は，前項で挙げた例からもあきらかである。

同時実行制御の対象データの範囲は，テーブルとする。本システムでは，テンポラリファイルを用いて処理を行っている。テーブルをレコードの集合と考えた場合，テンポラリファイルは指定された探索条件(の一部)を満足するその部分集合にあたる。１つのテーブルから複数のテンポラリファイル(部分集合)を得て探索条件の処理を行うため，同時実行制御の対象のデータの範囲をテーブルとする。範囲をテーブルにとることで，そのテーブルから得られるあらゆるテンポラリファイルがSQL文を実行する直前のテーブルデータを矛盾なく正確に反映していることが保証される。

4.2 ロックの種類

(1) テーブルロック

データベース中のテーブルを対象とする同時実行制御としてテーブルロックを用いる。操作するテーブルにロックをかけることで同時実行制御を行う。

本システムでは，次の２種類のテーブルロックを用いる。

・Rロック(Ｒｅａｄロック)

読み出しだしロック。テーブルを読み出す時に，このロックをかける。

・Wロック(Ｗｒｉｔｅロック)

書き込みロック。テーブルに書き込みを行う時に，このロックをかける。

ここで，RロックとWロックの関係について，以下の条件が成り立つ。

(a) ロックがかかっていないテーブルに対して，クライアントはRロック，Wロックをかけることができる。

(b) Rロックがかかっているテーブルに対して，クライアントはRロックをかけることができる。しかし，Wロックをかけることはできない。

(c) Wロックがかかっているテーブルに対して，クライアントはRロック，Wロックともかけることができない。

以上を表１にまとめる。

表１ ロックの性質

	－	R	W
－	○	○	○
R	○	○	×
W	○	×	×

－：ロックがかかっていない
R：Rロック
W：Wロック

○：ロック可能　×：ロック不可能

以上からわかるとおり，Rロックは重複してかけることができる。

(2) データベースロック

D-SQL の動作モードとして次の2種類ある。

(a) マルチユーザ・マルチクライアントモード

(b) シングルユーザ・シングルクライアントモード

データベースの再編成操作や保守作業時は動作モードをシングルユーザ・シングルクライアントモードにしてから行う必要がある。例えば，“reorganize”はデータベースの再編成を行うコマンドである。主データベースにあるテーブルすべてに対して，delete コマンドなどで消去されたテーブルレコードのゴミ集めを行う。非常に時間のかかる処理であり，その間クライアントを待ち行列で待たせておくのは適当でない。データベースを独占し，他のクライアントプロセスからの要求を受け付けないモードが必要になる。データベースロックは，データベースを対象とする排他制御である。このロックはデータベースをシングルユーザ・シングルクライアントモードで使用するためのロックである。

4.2 テーブルロッキングアルゴリズム

SQL コマンドの多くはユーザテーブルを操作する際に，その情報を知るためまたは，更新するためにまずシステムテーブルにアクセスする。従って，ユーザテーブルのロックに先立って，システムテーブルのロックが必要である。複数のシステムテーブルにロックをかける場合は，まずシステムテーブルのロックをすべてかけ終えてから，ユーザテーブルのロックをすべてかける。システムテーブルとユーザテーブルのロックのかけ方はまったく同一である。SQL コマンドの処理方法を分析した結果，必要なロッキングのパターンは次の2種類であることが判明した。

- 複数のテーブルにRロックをかける。
- 複数のテーブルにRロックをかけ，かつ，複数のテーブルにWロックをかける。

この2種類をシステムテーブル，ユーザテーブルそれぞれの場合に適用する。

以下に，これらのロックのかけ方を述べる。

(1) 複数のテーブルにRロックをかける。

待ち行列を用いる。クライアントは，SQL 操作文で使用される すべてのテーブルにRロックの要求を出す。ロックがかからなかった場合は，待ち行列に入りロックがかかるのを待つ。クライアントは，ロックを要求したすべてのテーブルにロックがかかるのを待って SQL 操作文の処理を開始する。

(2) 複数のテーブルにRロックをかけ，かつ，複数のテーブルにWロックをかける。

まず，Wロックをかけるテーブルのうちいずれか1つにWロックを要求する。ロックがかからなかった場合は，ロックがかかるまで待ち行列に入り待つ。このロックがかかるのを待って，Rロックも含めて残ったテーブルのロックを要求する。残ったロックの要求は，RPC 登録ルーチン lock2() によって行う。lock2() は，ロックをかけられなかった場合，待ち行列にならばない関数である。lock2() によって，ロックがかからないテーブルが1つでもあればそれまでにかけたロックをすべて解除する。このロック解除の要求は，クライアントがサーバに対して出す。以上の，動作をすべてのテーブルにロックがかかるまで繰り返す。

以上のようなロッキングのアルゴリズムで デッドロックが発生しないことが証明できるが，紙面の都合上省略する。

5．負荷分散の方針

サーバがデータベースを直接管理し，しかもクライアント・サーバ方式で運用される場合，常にサーバ側に過大な負荷をかけがちになる。このようなサーバ側の過大負荷は回避し，ネットワーク上の各ノードの負荷が偏らないようにしなければならない。すなわち，ネットワーク上に少数のデータベースサーバが存在し，これらに対して，多くのクライアントがデータベース操作を依頼したとしても遅滞することなく処理できるような，適切な負荷分散の方策を検討することは分散DBMSの場合には最も重要な課題の一つであるといえる。

一般にselectコマンド等に指定される条件節(where句)の処理には，大量のファイル操作に加えて，過酷なcpu処理が必要となる。従って，条件節の処理をすべてサーバで行わせた場合，クライアントの数が増大すれば，サーバ計算機に過大な負荷が集中し全体としてのスループットが極度に低下する。データベースが，複数の計算機に分散している場合には，分散先のサーバがそれぞれ自分が管理しているテーブルに関する検索を行うことになり，負荷分散をかなりの程度，達成できる。しかし，この場合でもクライアントとなるノードの数がサーバとなるノードの数をかなり上回る状況になれば，全体として円滑な操作・運用は望めない。UNIXネットワークのような同程度の能力を有する自立型のワークステーションが数多く存在するような環境下ではこのような状況による負荷集中の弊害は一層深刻になる。

サーバノードとクライアントノードの数の比を適正に保つために，可能ならばサーバノードの数を増やせば，このような負荷集中は避けられる。しかし，クライアントが発行したコマンドの条件節の中に指定されているテーブルが複数個であり，しかもそれらが複数のサーバノードに分散している場合には，条件節の最終結果を得るために大量のテーブルデータがネットワーク上を流れるためネットワークに過大な負荷をかけ，やはり全体としてのスループットが低下する。

また，各々のテーブルのデータ量も一般に均等ではなく，さらにデータベース中のテーブル集合を“論理的まとまり”の観点から分割した場合と，負荷分散の見地にたって物理的に分割した場合とでは必ずしも分割が一致しない。したがって，サーバの数を増やして，負荷集中を回避するという方策にも限度があると考えられる。

以上のような考察から，ネットワーク上の分散データベース管理システムを設計する上での主な要点は以下の2点であるということができる。

(i) サーバノード(群)への負荷集中の軽減

(ii) ネットワーク通信量の軽減

以上の2点を共に満足する解決法を見つけだすことは非常に困難であるし，データベースの運用形態も多様であることを考えると唯一絶対的な方策というようなものはあり得ない。

D-SQLでは(i)の点を重視した上で，可能な限りネットワーク通信量を軽減するような方策とする。そのために，データベースの中心的な処理である条件探索操作について，負荷の比較的軽いものだけをサーバで行い，テーブルの結合操作(join)などを含む負荷の重い条件探索操作はクライアント側で行うこととする。これによって，サーバノードに対する負荷は，かなりクライアント側に移動することとなり，システム全体としてのスループットを向上させることができる。

6．結　　言

UNIX LAN 上の関係型分散 DBMS の設計と構築について述べた。現在 D-SQL のプロトタイプが 2，3 の計算機上で動作している。今後，本システムをプラットホームとしてデータベースシステムや分散システムそのもの，あるいはそれらに基づいたソフトウエア開発統合環境に関する研究を行う予定である。

参考文献

1) C.J.Date,（芝野監訳),'標準 SQL：第 2 版'，トッパン，1990.
2) 植村俊亮，'データベースシステムの基礎'，オーム社，1979.
3) 都司達夫，'UNIX 上における ISAM パッケージの実現'，福井大学情報処理センターニュース(NETWORK)，Vol.3，No.1，115-133，1989.
4) W.R.Stevens，UNIX Network Programming: Prentice Hall，Englewood Cliffs，NJ，1990.
5) 村井，砂原，横手，'UNIX ワークステーション'，アスキー，1987.
6) E.Ozkarahan，"Database Management: Concepts，Design，and Practice"，Prentice Hall，Englewood Cliffs，New Jersey 07632，1990.
7) U.Rodgers，"UNIX Database Management Systems"，Prentice-Hall Hall，Englewood Cliffs，New Jersey 07632，1990.
8) C.J.Date，"A Guide to Ingress"，Addison-Wesley Publishing Company，1987.
9) Hewlett Pakard Co.，"Programming and Protocols for NFS Services"，Manual Part Number - 50969-90010.
10) 都司達夫，UNIX 上における分散 DBMS の構築(II)－(IX)，インターフェース，1991年 7 月号から1992年 3 月号まで連載.